NGINX Conf 2018

The official event for all things NGINX

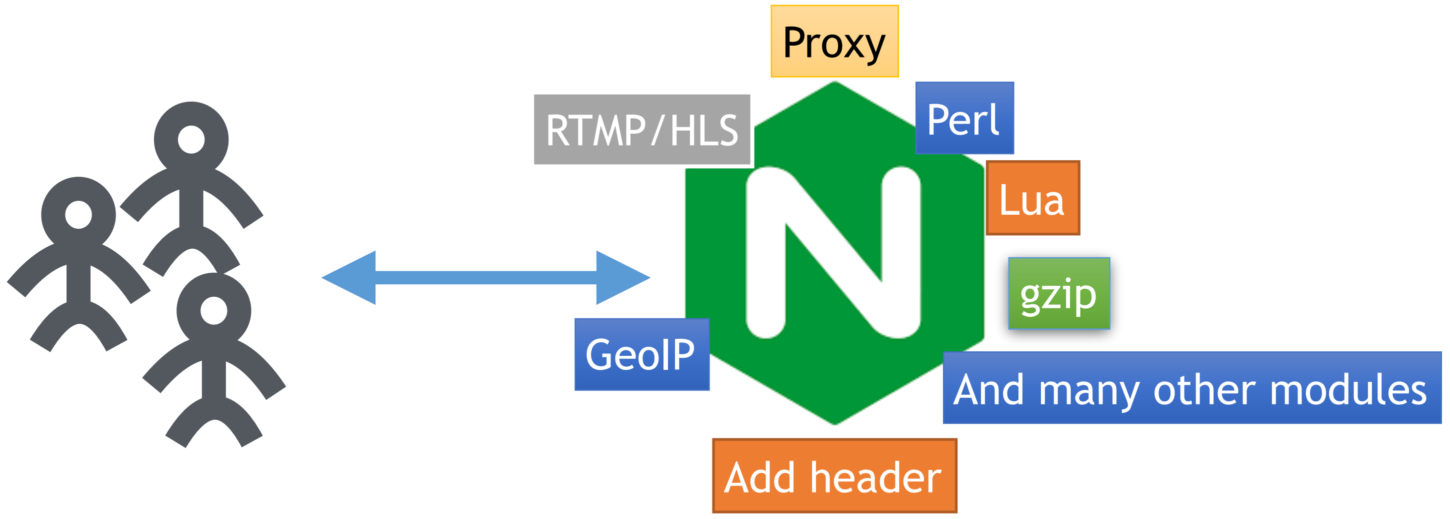
Why make your own NGINX modules? Theory and Practice

Vasiliy Soshnikov, Mail.Ru Group
vasiliy.soshnikov@gmail.com

Agenda

Why make your own modules?

- To add new features
- To solve problems and business issues



Internet analytics & statistics

- To set special cookies
- To collect special logs.

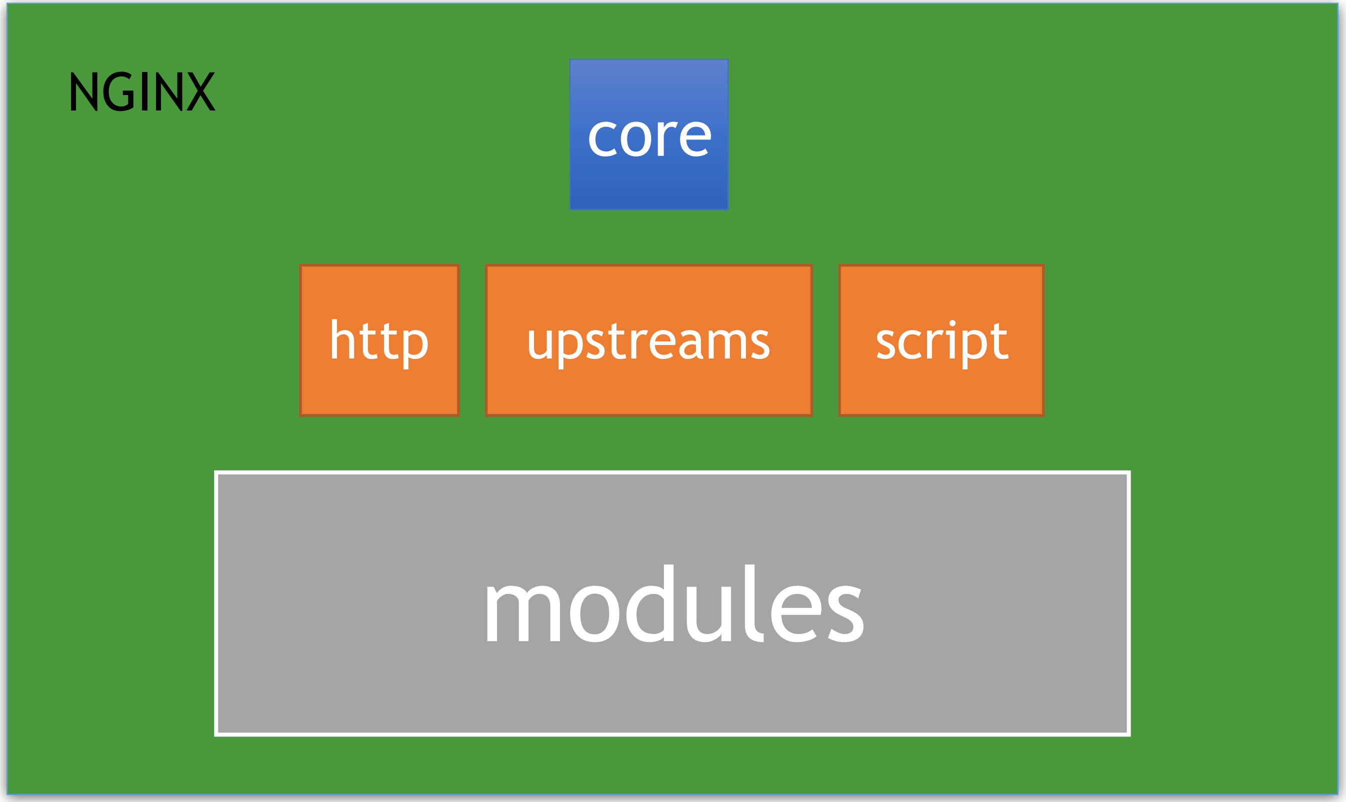
Advertisement - Ads

- To return banners in real-time (typical internet ***ads***)
- To collect ***ads*** statistics.

Special Proxies

- To convert HTTP to X (in present case in MsgPack) protocol
- To load balancing
- To have failover
- More features: http://nginx.org/en/docs/http/ngx_http_upstream_module.html.

Foreword. NGINX Architecture

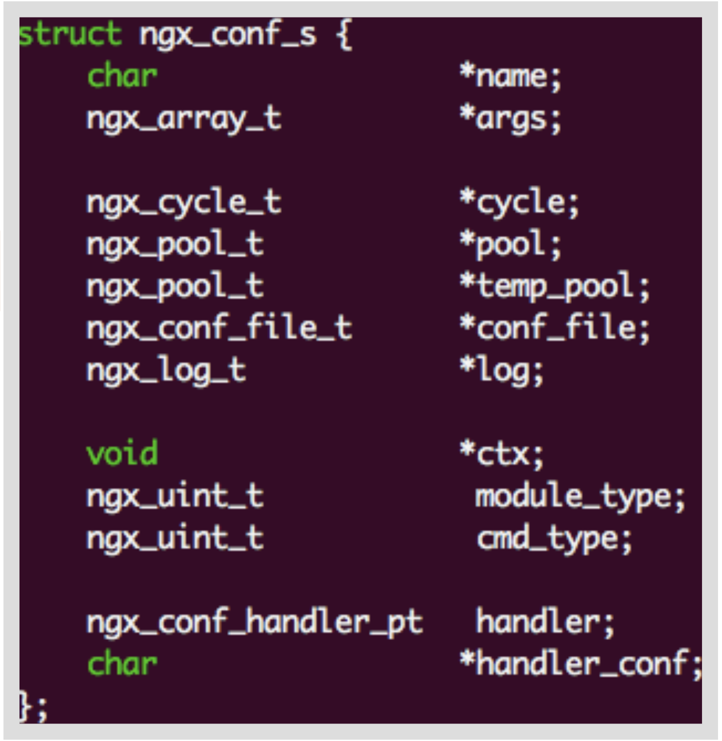


Foreword. NGINX Memory model

- Pool allocation - means you don't need to use "free()" functions
- You have to choose a right pool!
- Try to use NGINX memory functions which allows to work with NGINX pools
- Avoid external functions like “malloc()”.

Sources: nginx/src/core/ngx_palloc.h

A reference to pool >



```
struct ngx_conf_s {
    char                    *name;
    ngx_array_t             *args;

    ngx_cycle_t             *cycle;
    ngx_pool_t              *pool;
    ngx_pool_t              *temp_pool;
    ngx_conf_file_t         *conf_file;
    ngx_log_t               *log;

    void                    *ctx;
    ngx_uint_t               module_type;
    ngx_uint_t               cmd_type;

    ngx_conf_handler_pt      handler;
    char                    *handler_conf;
};
```

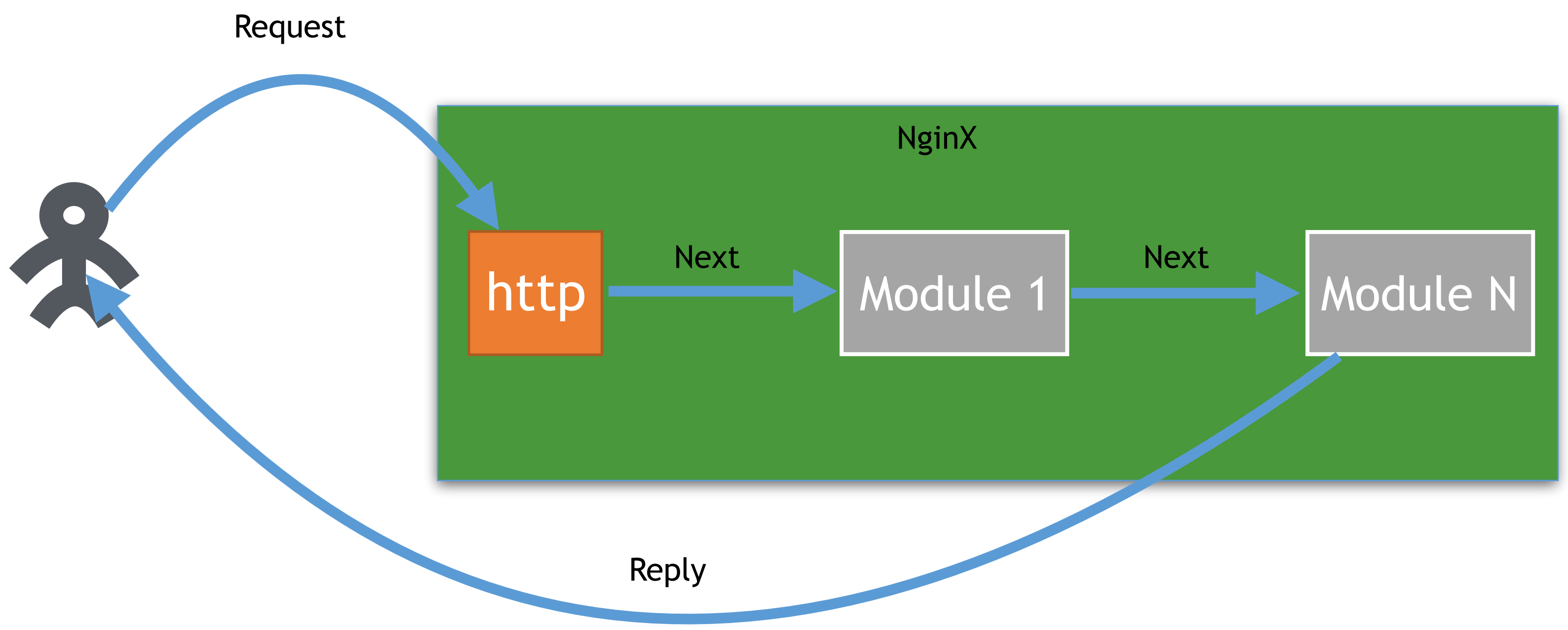
Foreword. NGINX is a framework

- Data structures: B-tree, Hash, Array, List, Radix tree etc.
- OS API: File I/O, Shared Memory etc.
- Event-driven state machine: polling and timers.

Sources: nginx/src/core/, nginx/src/http/*, nginx/src/event/*, nginx/src/os/**

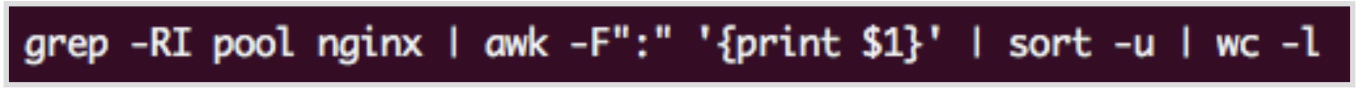
Theory

Chain of Responsibility



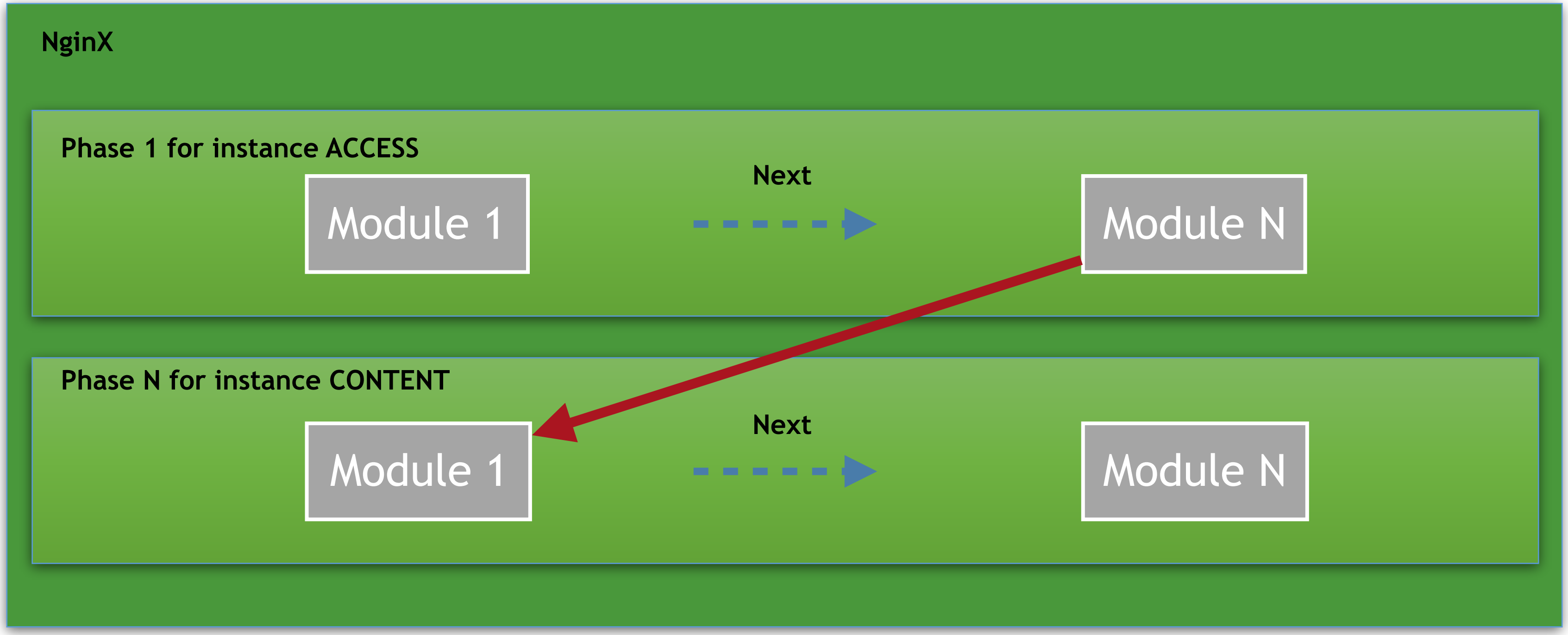
Chain of Responsibility

Analogy pattern is (bash-script):



```
grep -RI pool nginx | awk -F":" '{print $1}' | sort -u | wc -l
```

Phases


NginX
Phase 1 for instance ACCESS
Module 1
Next
Module N
Phase N for instance CONTENT
Module 1
Next
Module N

Module types

- Phase handlers
- Header and Body filters
- Proxies
- Load balancers

Practice

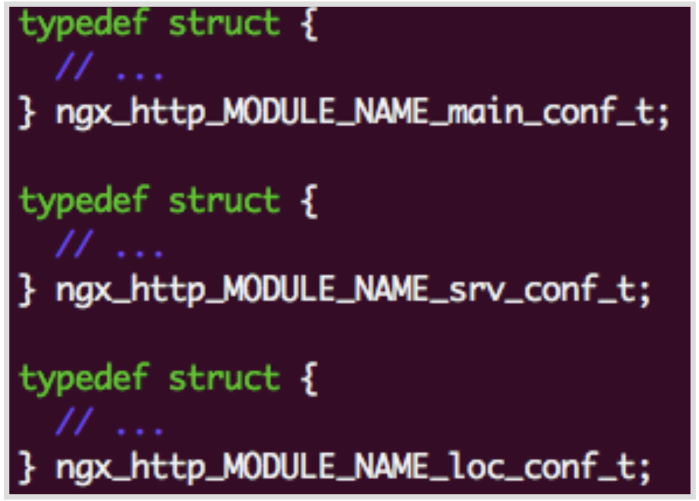
Foreword

1. A module begins with configuration.

2. Highly recommended naming policy is:

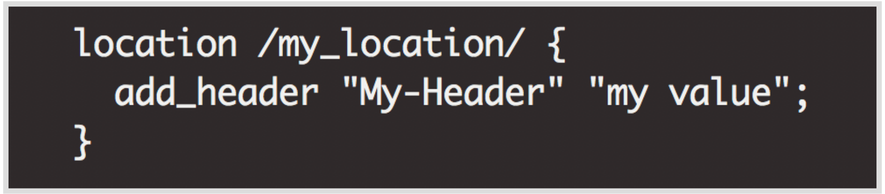
ngx_http_NAME_{main, srv, loc}

where NAME is a module name.

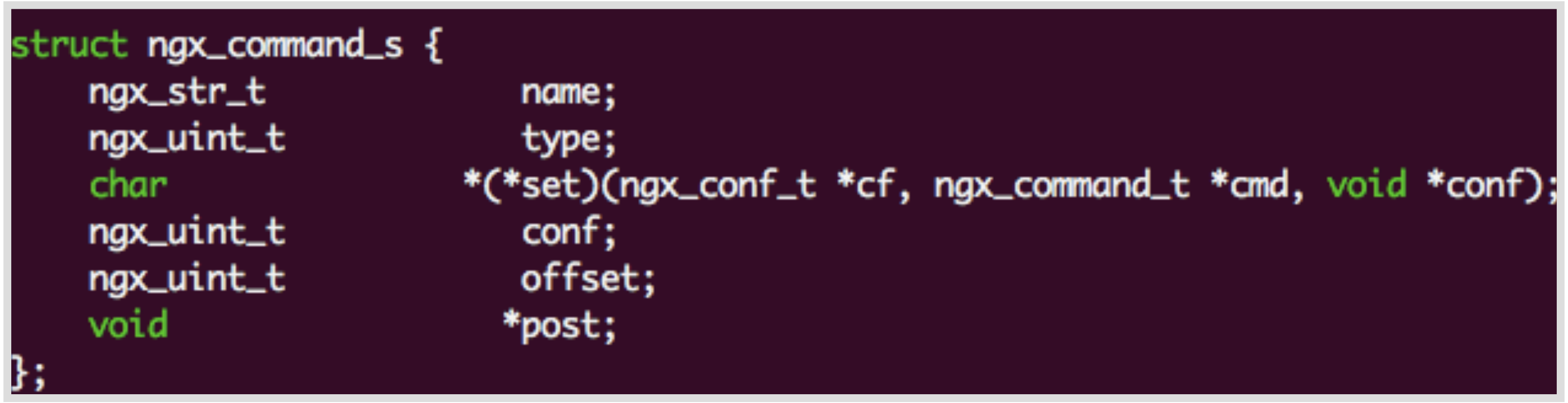


```
typedef struct {
  // ...
} ngx_http_MODULE_NAME_main_conf_t;

typedef struct {
  // ...
} ngx_http_MODULE_NAME_srv_conf_t;

typedef struct {
  // ...
} ngx_http_MODULE_NAME_loc_conf_t;
```

NGINX’s configuration directives



```
location /my_location/ {
  add_header "My-Header" "my value";
}
```

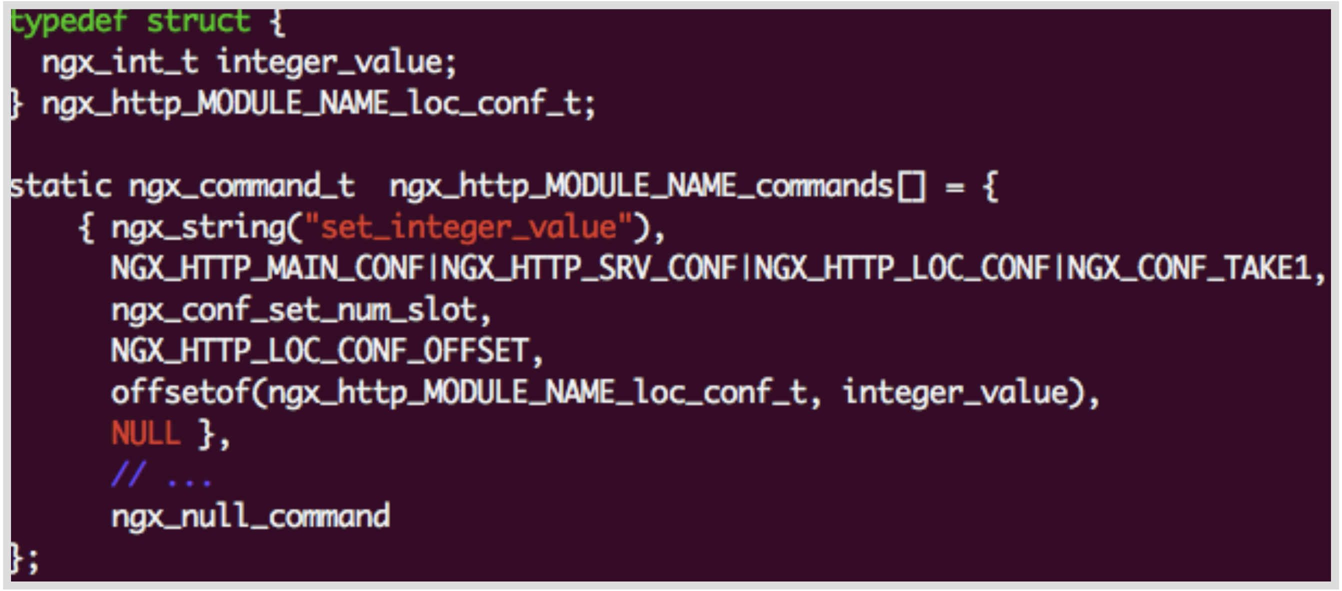


```
struct ngx_command_s {
    ngx_str_t             name;
    ngx_uint_t            type;
    char               *(*set)(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
    ngx_uint_t            conf;
    ngx_uint_t            offset;
    void                 *post;
};
```

Sources: nginx/src/core/ngx_conf_file.{h, c}

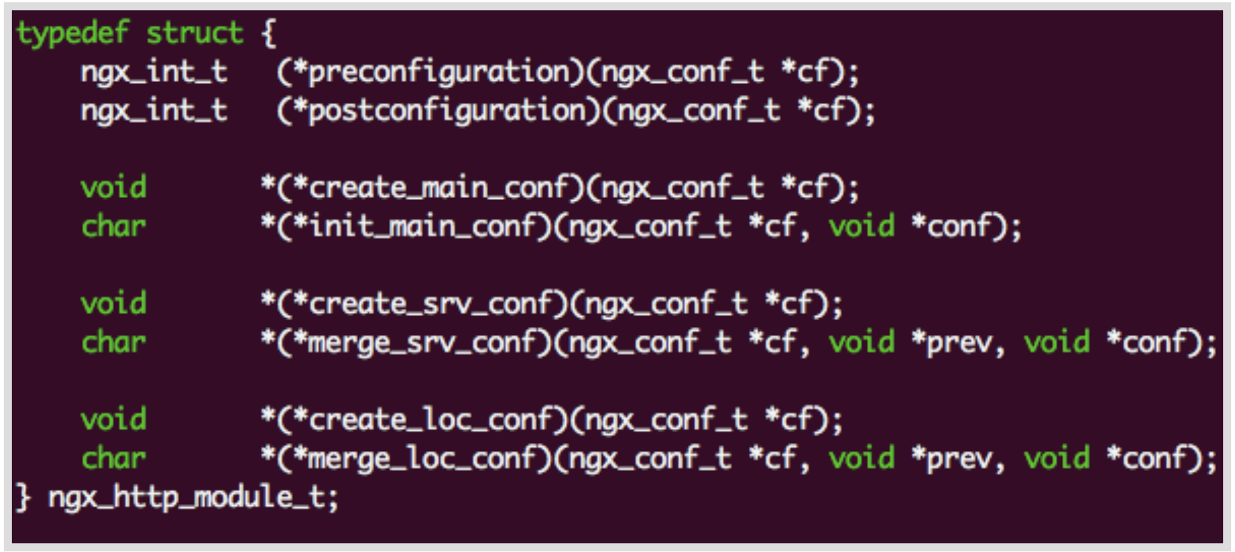
NGINX's configuration directives

Example:



```
typedef struct {
  ngx_int_t integer_value;
} ngx_http_MODULE_NAME_loc_conf_t;

static ngx_command_t  ngx_http_MODULE_NAME_commands[] = {
    { ngx_string("set_integer_value"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_MODULE_NAME_loc_conf_t, integer_value),
      NULL },
      // ...
      ngx_null_command
};
```

Add a new module

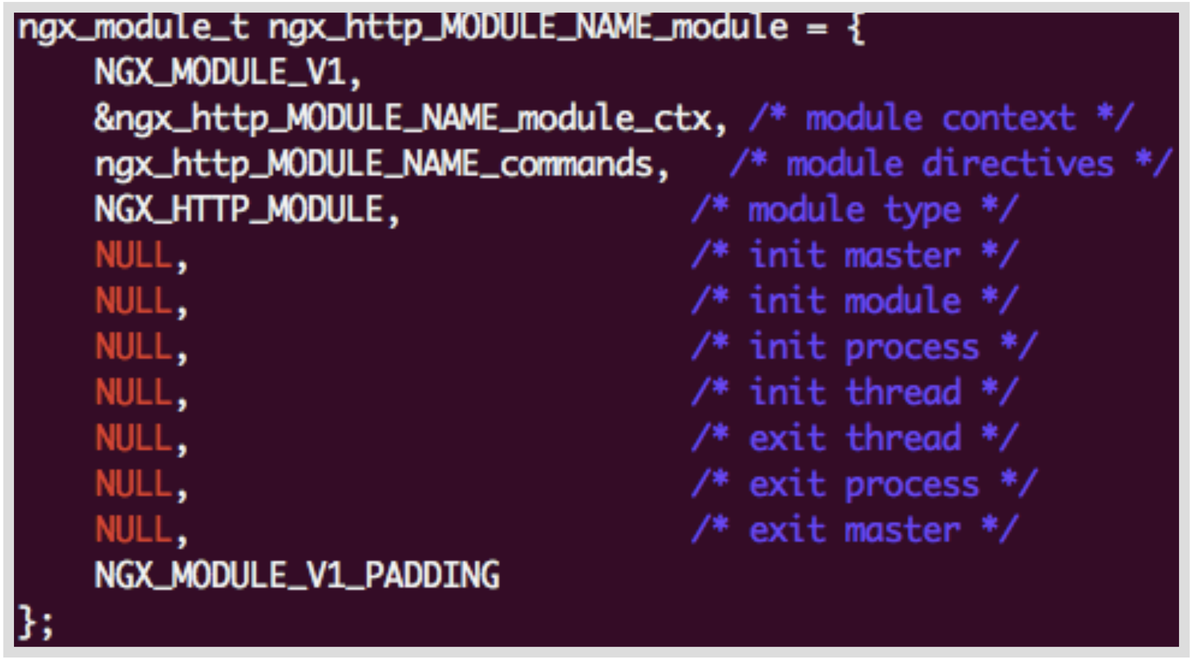


```
typedef struct {
    ngx_int_t   (*preconfiguration)(ngx_conf_t *cf);
    ngx_int_t   (*postconfiguration)(ngx_conf_t *cf);

    void       *(*create_main_conf)(ngx_conf_t *cf);
    char       *(*init_main_conf)(ngx_conf_t *cf, void *conf);

    void       *(*create_srv_conf)(ngx_conf_t *cf);
    char       *(*merge_srv_conf)(ngx_conf_t *cf, void *prev, void *conf);

    void       *(*create_loc_conf)(ngx_conf_t *cf);
    char       *(*merge_loc_conf)(ngx_conf_t *cf, void *prev, void *conf);
} ngx_http_module_t;
```

Sources: nginx/src/http/ngx_http_config.h

Add a new module

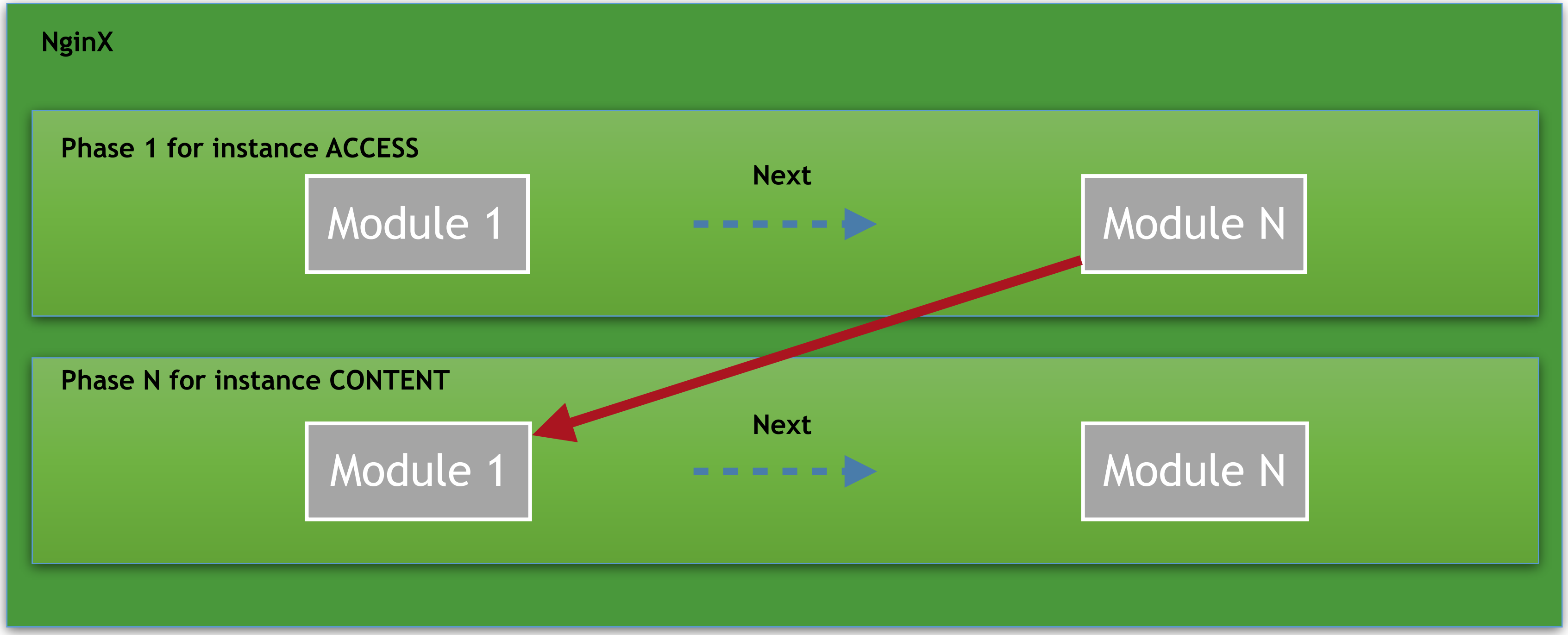
Example:



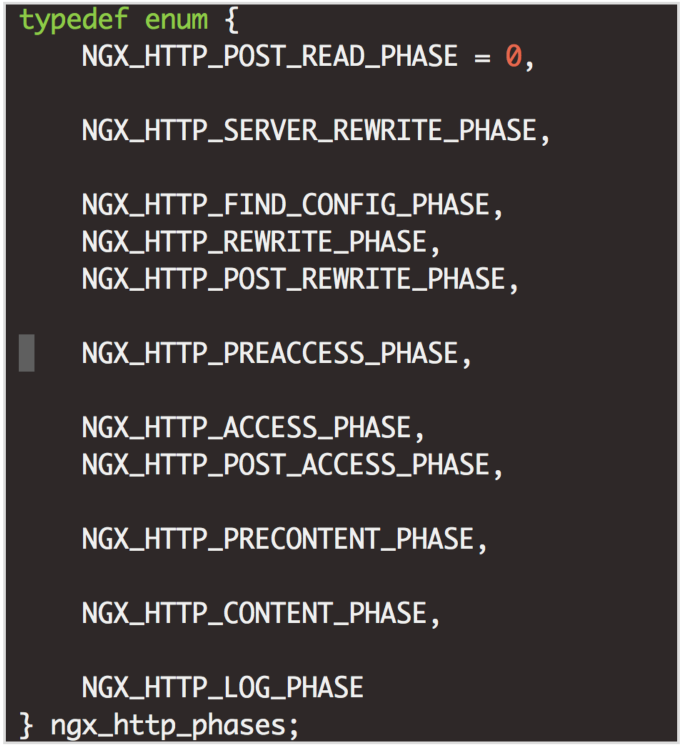
```
ngx_module_t ngx_http_MODULE_NAME_module = {
    NGX_MODULE_V1,
    &ngx_http_MODULE_NAME_module_ctx, /* module context */
    ngx_http_MODULE_NAME_commands,   /* module directives */
    NGX_HTTP_MODULE,                /* module type */
    NULL,                           /* init master */
    NULL,                           /* init module */
    NULL,                           /* init process */
    NULL,                           /* init thread */
    NULL,                           /* exit thread */
    NULL,                           /* exit process */
    NULL,                           /* exit master */
    NGX_MODULE_V1_PADDING
};
```

Phase handlers

Phases


NginX
Phase 1 for instance ACCESS
Module 1
Next
Module N
Phase N for instance CONTENT
Module 1
Next
Module N

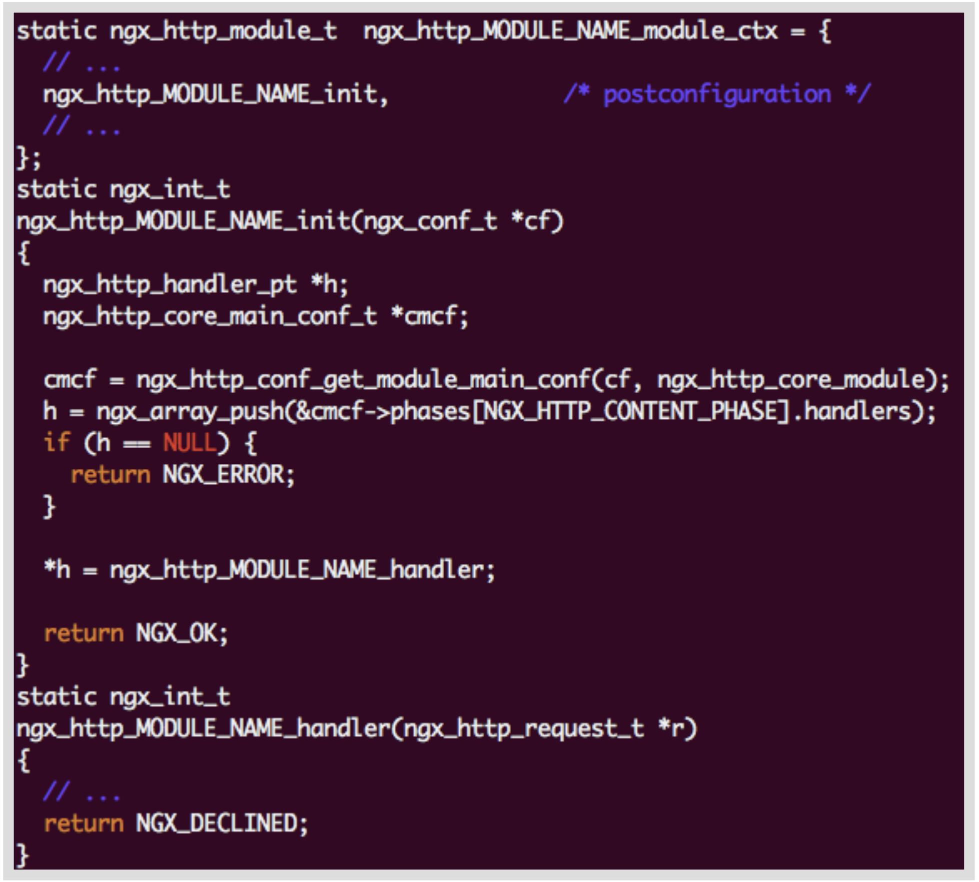
Phases



```
typedef enum {
    NGX_HTTP_POST_READ_PHASE = 0,

    NGX_HTTP_SERVER_REWRITE_PHASE,

    NGX_HTTP_FIND_CONFIG_PHASE,
    NGX_HTTP_REWRITE_PHASE,
    NGX_HTTP_POST_REWRITE_PHASE,

    NGX_HTTP_PREACCESS_PHASE,

    NGX_HTTP_ACCESS_PHASE,
    NGX_HTTP_POST_ACCESS_PHASE,

    NGX_HTTP_PRECONTENT_PHASE,

    NGX_HTTP_CONTENT_PHASE,

    NGX_HTTP_LOG_PHASE
} ngx_http_phases;
```

Sources: nginx/src/http/ngx_http_core_module.h

Set >



```
static ngx_http_module_t  ngx_http_MODULE_NAME_module_ctx = {
  // ...
  ngx_http_MODULE_NAME_init,             /* postconfiguration */
  // ...
};
static ngx_int_t
ngx_http_MODULE_NAME_init(ngx_conf_t *cf)
{
  ngx_http_handler_pt *h;
  ngx_http_core_main_conf_t *cmcf;

  cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
  h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
  if (h == NULL) {
    return NGX_ERROR;
  }

  *h = ngx_http_MODULE_NAME_handler;

  return NGX_OK;
}
static ngx_int_t
ngx_http_MODULE_NAME_handler(ngx_http_request_t *r)
{
  // ...
  return NGX_DECLINED;
}
```

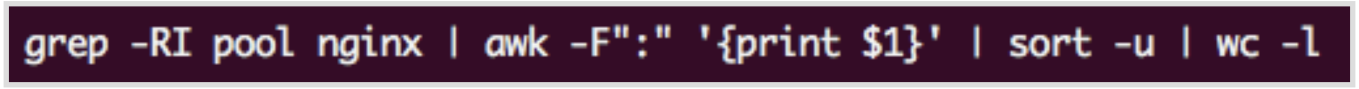
< Add handler

Impl.>

Header and Body filters

Chain of Responsibility

Analogy pattern is (bash-script):

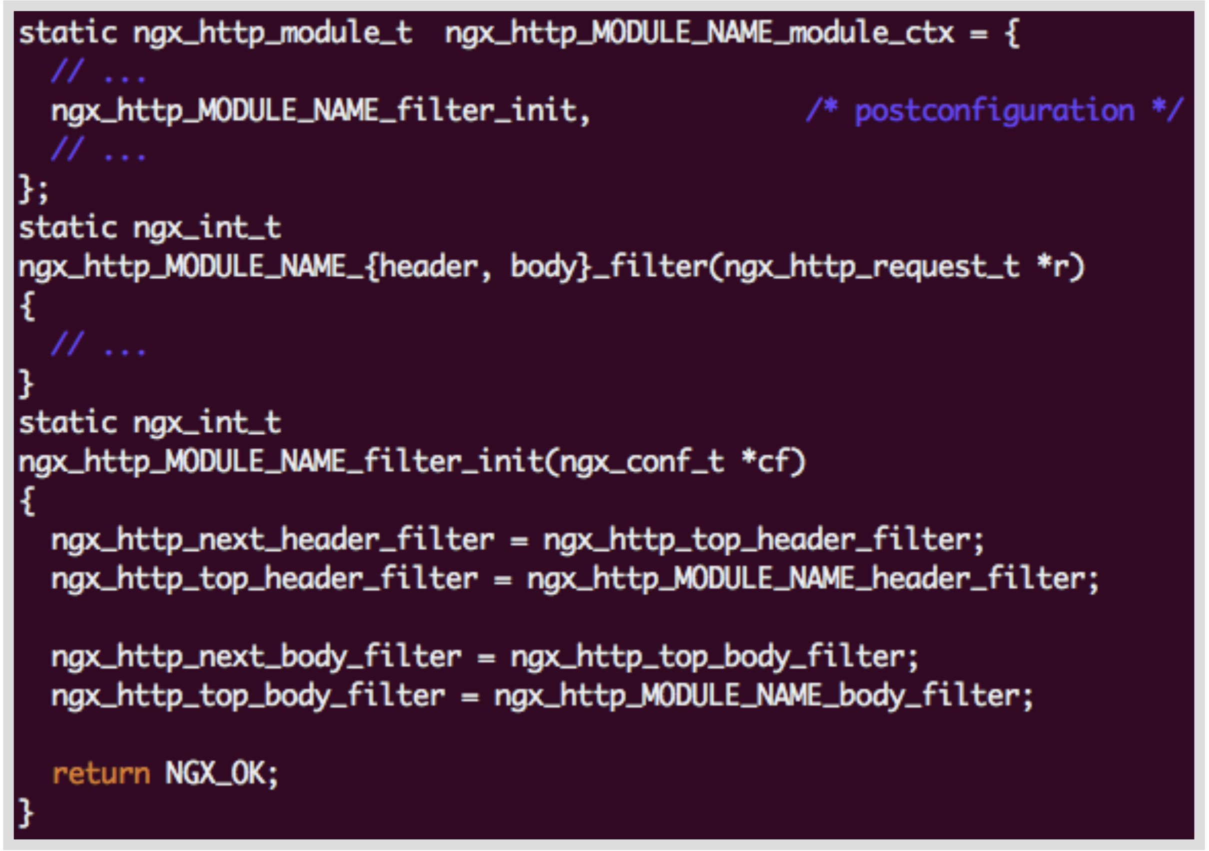


```
grep -RI pool nginx | awk -F":" '{print $1}' | sort -u | wc -l
```

Set >

Header filter >

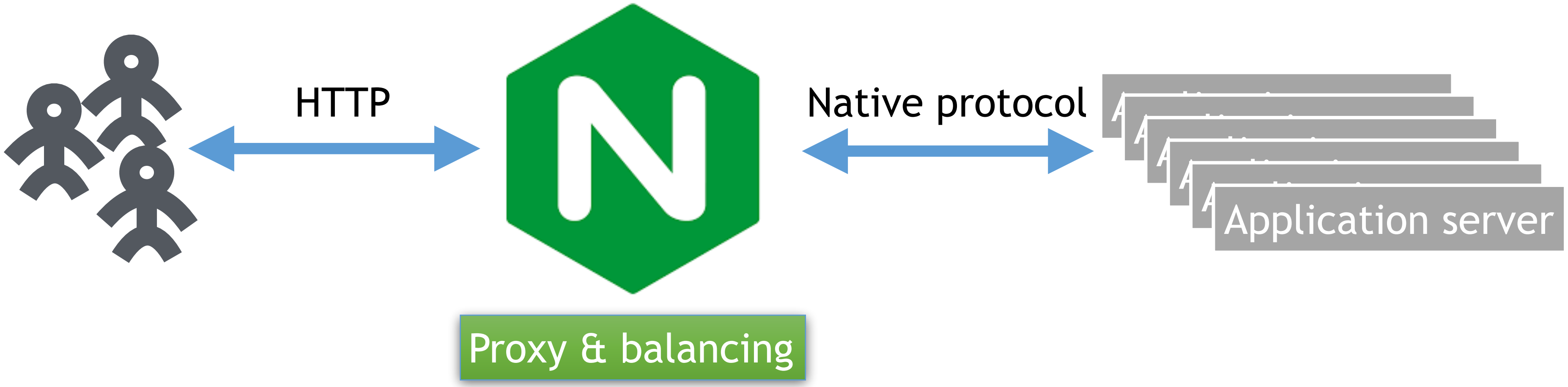
Body filter >



```
static ngx_http_module_t  ngx_http_MODULE_NAME_module_ctx = {
  // ...
  ngx_http_MODULE_NAME_filter_init,             /* postconfiguration */
  // ...
};
static ngx_int_t
ngx_http_MODULE_NAME_{header, body}_filter(ngx_http_request_t *r)
{
  // ...
}
static ngx_int_t
ngx_http_MODULE_NAME_filter_init(ngx_conf_t *cf)
{
  ngx_http_next_header_filter = ngx_http_top_header_filter;
  ngx_http_top_header_filter = ngx_http_MODULE_NAME_header_filter;

  ngx_http_next_body_filter = ngx_http_top_body_filter;
  ngx_http_top_body_filter = ngx_http_MODULE_NAME_body_filter;

  return NGX_OK;
}
```

Proxies

Anatomy

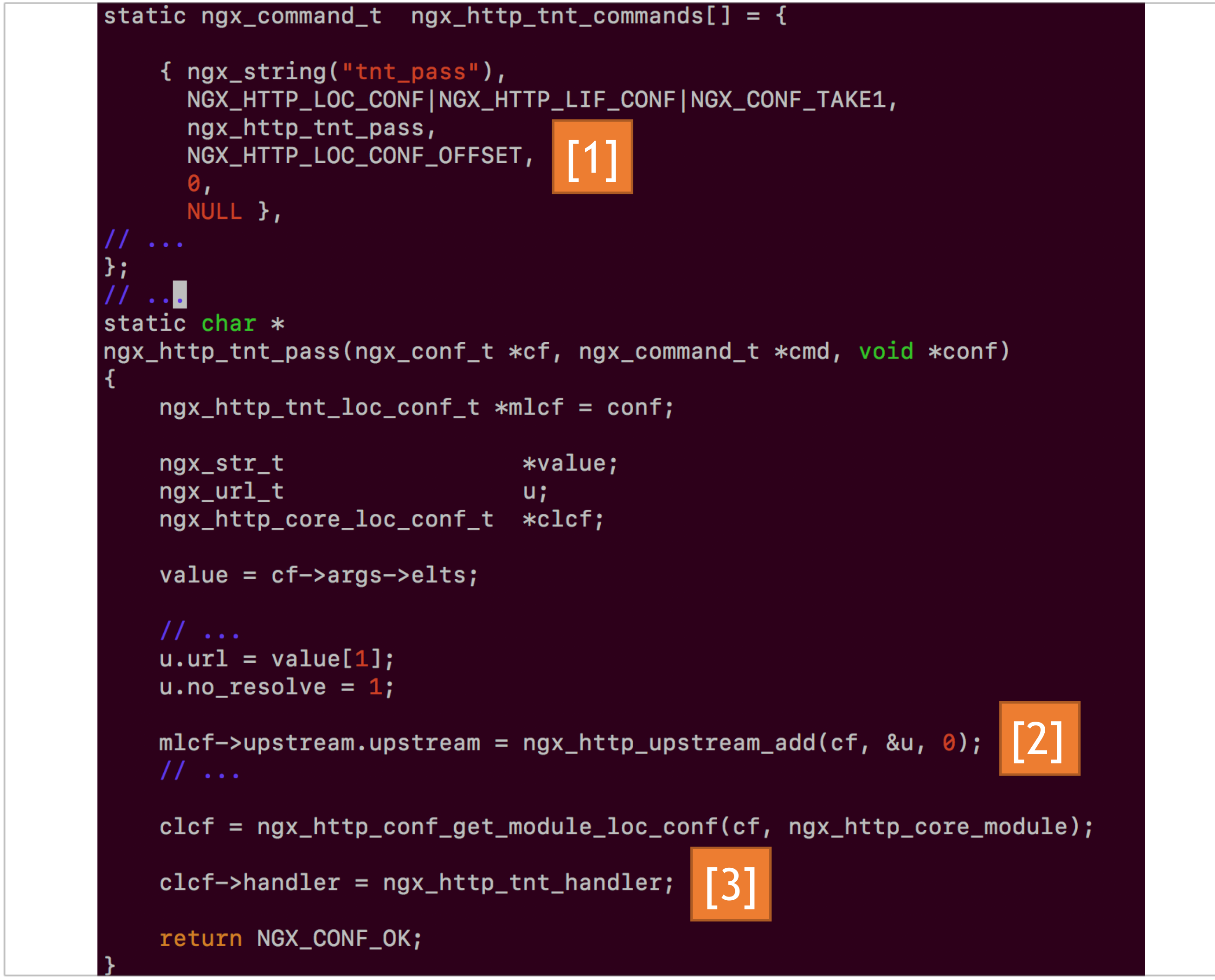


Registration

[1] Add a new directive.

[2] Add a new upstream.

[3] Add a new handler to the added upstream.


```
static ngx_command_t  ngx_http_tnt_commands[] = {

    { ngx_string("tnt_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_tnt_pass,
      NGX_HTTP_LOC_CONF_OFFSET,  [1]
      0,
      NULL },
// ...
};
// ...
static char *
ngx_http_tnt_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_tnt_loc_conf_t *mlcf = conf;

    ngx_str_t                 *value;
    ngx_url_t                 u;
    ngx_http_core_loc_conf_t  *clcf;

    value = cf->args->elts;

    // ...
    u.url = value[1];
    u.no_resolve = 1;

    mlcf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);  [2]
    // ...

    clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);

    clcf->handler = ngx_http_tnt_handler;  [3]

    return NGX_CONF_OK;
}
```

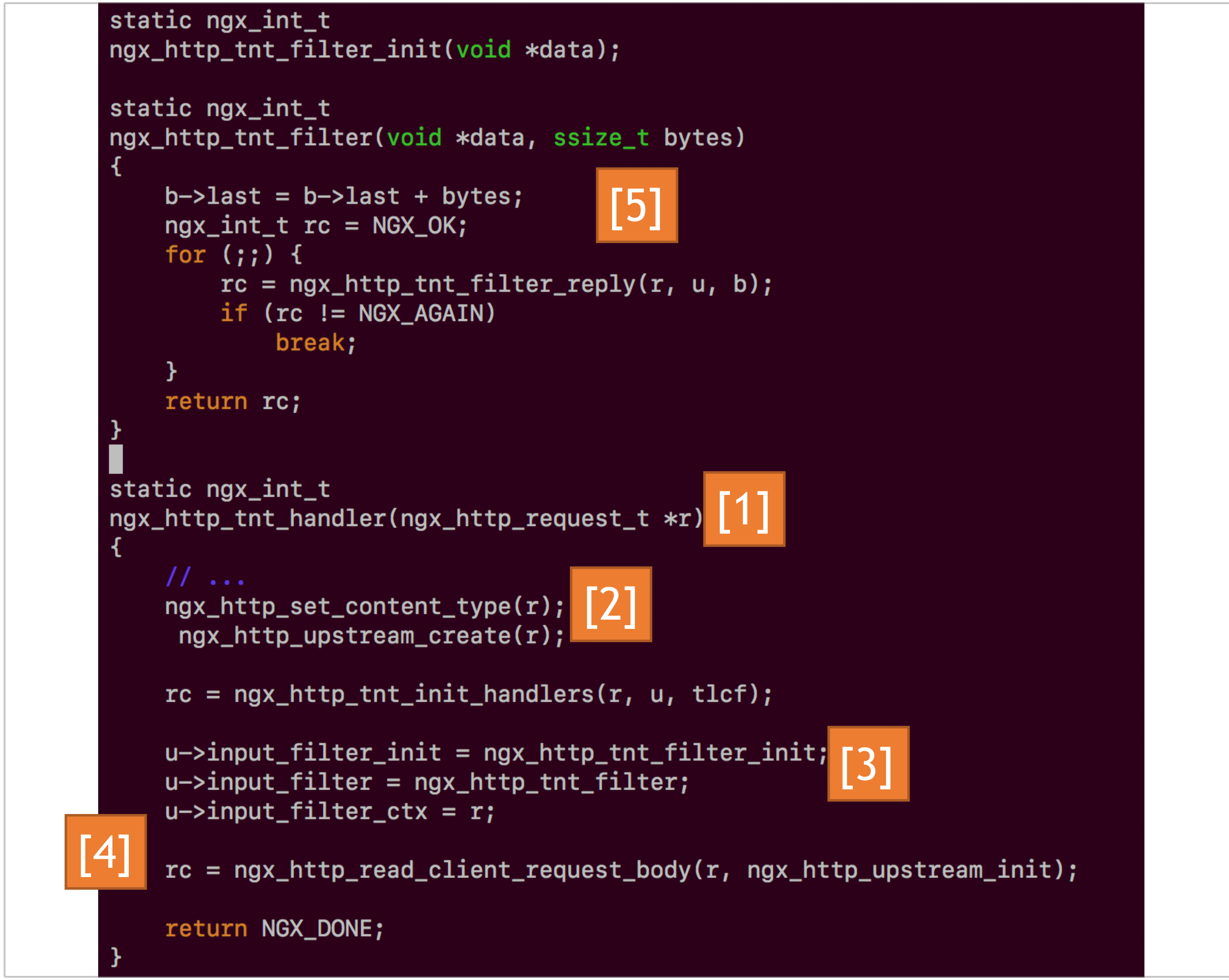
Content handling

[1] Invoked on request.

[2] Create Upstream & Downstream (or getting from KeepAlive module).

[3, 4] Add handlers & filters (also it converts the request to the backend request).

[5] Read backend reply.



```
static ngx_int_t
ngx_http_tnt_filter_init(void *data);

static ngx_int_t
ngx_http_tnt_filter(void *data, ssize_t bytes)
{
    b->last = b->last + bytes;
    ngx_int_t rc = NGX_OK;
    for (;;) {
        rc = ngx_http_tnt_filter_reply(r, u, b);
        if (rc != NGX_AGAIN)
            break;
    }
    return rc;
}

static ngx_int_t
ngx_http_tnt_handler(ngx_http_request_t *r)
{
    // ...
    ngx_http_set_content_type(r);
     ngx_http_upstream_create(r);

    rc = ngx_http_tnt_init_handlers(r, u, tlcf);

    u->input_filter_init = ngx_http_tnt_filter_init;
    u->input_filter = ngx_http_tnt_filter;
    u->input_filter_ctx = r;

    rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init);

    return NGX_DONE;
}
```

References

- https://github.com/dedok/nginx-tutorials — tutorials
- https://github.com/tarantool/nginx_upstream_module — an example of Proxy (a real project)
- https://www.nginx.com/resources/wiki/modules/ — about NGINX modules
- http://nginx.org/en/docs/dev/development_guide.html — a development guide
- nginx.org — NGINX web site.

NGINX

Thank you!

Contact information:
vasiliy.soshnikov@gmail.com
GitHub: @dedok